Laborator si seminar

Programare in Java si software matematic

Adrian Ichimescu

Adrianichimescu@gmail.com

Object-Oriented Programming Concepts

- Object

  An object is a software bundle of related state and behavior.
- Class

  A class is a blueprint or prototype from which objects are created.
- Inheritance

  classes inherit state and behavior from their superclasses
- Interface

  interface is a contract between a class and the outside world
- Package

  is a namespace for organizing classes and interfaces in a logical manner.

Object-Oriented Programming Benefits

- Object

  An object is a software bundle of related state and behavior.

  State = fields

  Behaviour = Methods

OOP benefits:

Modularity: The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.

Information-hiding: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.

Code re-use: If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.

Pluggability and debugging ease: If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace it, not the entire machine.

Exercise

Bicycle is an *instance* of the *class of objects* known as bicycles

```
class Bicycle {

  int cadence = 0;

   int speed = 0;

   int gear = 1;

void changeCadence(int newValue) {

       cadence = newValue;   }

void changeGear(int newValue) {

       gear = newValue;   }

void speedUp(int increment) {

       speed = speed + increment;    }

    void applyBrakes(int decrement) {

       speed = speed - decrement;   }

    void printStates() {

       System.out.println("cadence:" + cadence + " speed:" + speed + " gear:" + gear);   }

}
```

The fields cadence, speed, and gear represent the object's state, and the methods (changeCadence, changeGear, speedUp etc.) define its interaction with the outside world

Exercise

```
class BicycleDemo {
  public static void main(String[] args) {
     // Create two different  Bicycle objects
     Bicycle bike1 = new Bicycle();
     Bicycle bike2 = new Bicycle();
     // Invoke methods on
     // those objects
     bike1.changeCadence(50);
     bike1.speedUp(10);
     bike1.changeGear(2);
     bike1.printStates();
      bike2.changeCadence(50);
     bike2.speedUp(10);
     bike2.changeGear(2);
     bike2.changeCadence(40);
     bike2.speedUp(10);
     bike2.changeGear(3);
     bike2.printStates();
  }
}
```

the **Bicycle** class does not contain a **main** method. That's because it's not a complete application; it's just the prototype for bicycles that might be used in an application. The responsibility of creating and using new Bicycle objects belongs to some other class in our application called **BicycleDemo**.

Exercise

1) What is the output running BicycleDemo ?

2) Create new classes for a real-world Automobile object. Refer to the Bicycle class if you forget the required syntax.

Inheritance

Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes.

A subclass extends the common characteristics (states) and behavior (methods) of its superclass using keyword ‘**extends**’

```
class MountainBike extends Bicycle {

   // new fields and methods defining
   // a mountain bike would go here

}
```

Interface

- Class methods are the external way of interacting with class and its objects
- Interface commonly is the group of methods with empty bodies

```
interface Bicycle {
   void changeCadence(int newValue);
   void changeGear(int newValue);
   void speedUp(int increment);
   void applyBrakes(int decrement);
}
```

Interface

If a class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

```
class FSABicycle implements Bicycle {
   int cadence = 0;
   int speed = 0;
   int gear = 1;
  // The compiler will now require that methods
  // changeCadence, changeGear, speedUp, and applyBrakes
  // all be implemented. Compilation will fail if those
  // methods are missing from this class.
public void changeCadence(int newValue) {
       cadence = newValue;}
public void changeGear(int newValue) {
       gear = newValue;}
public void speedUp(int increment) {
       speed = speed + increment;}
public void applyBrakes(int decrement) {
       speed = speed - decrement;}
public void printStates() {
       System.out.println("cadence:" + cadence + " speed:" + speed + " gear:" + gear);}
}
```

Exercise

- Write an application that create an object of FSABicycle class and prints its states

Exercise

- For the newly created **Automobile class**, create an **Interface** that defines its behavior, then require your class to implement it. Omit one or two methods and try compiling. What does the error look like?

Package

- A package is a namespace that organizes a set of related classes and interfaces.
- Java API is a collection of packages like String, Socket, File,…

https://docs.oracle.com/javase/8/docs/api/index.html

Exercise

- Play with importing previous package and call its class methods
- 3 importing options available:
  - The *JOptionPane* class is in the *swing* package, which is located in the *javax* package. The *wildcard* character (*) is used to specify that all classes with that package are available to your program. This is the most common programming style.

    import javax.swing.; // Make all classes visible althow only one is used.*
  - Classes can be specified explicitly on *import* instead of using the wildcard character.

    import javax.swing.JOptionPane; // Make a single class visible.
  - We can the fully qualified class name without an import.

    javax.swing.JOptionPane.showMessageDialog(null, "Hi");